(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 July 2003 (17.07.2003)

PCT

(10) International Publication Number
WO 03/058447 A2

(51) International Patent Classification[7]: G06F 9/46

(21) International Application Number: PCT/US02/39786

(22) International Filing Date:
11 December 2002 (11.12.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/039,579 31 December 2001 (31.12.2001) US

(71) Applicant: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventors: **MARR, Deborah**; 2564 NW Pettygrove Street, Portland, OR 97210 (US). **RODGERS, Scott**; 452 SW Brookwood Avenue, Hillsboro, OR 97123 (US). **HILL, David**; 37000 SW Goddard Road, Cornelius, OR 97113 (US). **KAUSHIK, Shivnandan**; 15417 NW Blakely Lane, Portland, OR 97229 (US). **CROSSLAND, James**; 10744 NW Davidson Rad, Banks, OR 97106 (US). **KOUFATY, David**; 6030 NW 163rd Place, Portland, OR 97229 (US).

(74) Agents: **MALLIE, Michael, J.** et al.; Blakely, Sokoloff, Taylor & Zafman, 7th Floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 03/058447 A2

(54) Title: A METHOD AND APPARATUS FOR SUSPENDING EXECUTION OF A THREAD UNTIL A SPECIFIED MEMORY ACCESS OCCURS

(57) Abstract: Techniques for suspending execution of a thread until a specified memory access occurs. In one embodiment, a processor includes multiple execution units capable of executing multiple threads. A first thread includes an instruction that specifies a monitor address. Suspend logic suspends execution of the first thread, and a monitor causes resumption of the first thread in response to an access to the specified monitor address.

A METHOD AND APPARATUS FOR SUSPENDING EXECUTION OF A THREAD UNTIL A SPECIFIED MEMORY ACCESS OCCURS

RELATED APPLICATIONS

[0001] This application is related to Application Serial Number __/___,___, entitled "Suspending Execution of a Thread in a Multi-threaded Processor"; Application Serial Number __/___,___, entitled "Coherency Techniques for Suspending Execution of a Thread Until a Specified Memory Access Occurs"; Application Serial Number __/___,___, entitled "Instruction Sequences for Suspending Execution of a Thread Until a Specified Memory Access Occurs" all filed on the same date as the present application.

BACKGROUND

1. Field

[0002] The present disclosure pertains to the field of processors. More particularly, the present disclosure pertains to multi-threaded processors and techniques for temporarily suspending the processing of one thread in a multi-threaded processor.

2. Description of Related Art

[0003] A multi-threaded processor is capable of processing multiple different instruction sequences concurrently. A primary motivating factor driving execution of multiple instruction streams within a single processor is the resulting improvement in processor utilization. Highly parallel architectures have developed over the years, but it is often difficult to extract sufficient parallelism from a single stream of instructions to utilize the multiple execution units. Simultaneous multi-threading processors allow multiple instruction streams to execute concurrently in the different execution resources in an attempt to better utilize those resources. Multi-threading can be particularly advantageous

for programs that encounter high latency delays or which often wait for events to occur. When one thread is waiting for a high latency task to complete or for a particular event, a different thread may be processed.

[0004] Many different techniques have been proposed to control when a processor switches between threads. For example, some processors detect particular long latency events such as L2 cache misses and switch threads in response to these detected long latency events. While detection of such long latency events may be effective in some circumstances, such event detection is unlikely to detect all points at which it may be efficient to switch threads. In particular, event based thread switching may fail to detect points in a program where delays are intended by the programmer.

[0005] In fact, often, the programmer is in the best position to determine when it would be efficient to switch threads to avoid wasteful spin-wait loops or other resource-consuming delay techniques. Thus, allowing programs to control thread switching may enable programs to operate more efficiently. Explicit program instructions that affect thread selection may be advantageous to this end. For example, a "Pause" instruction is described in US Patent Application No. 09/489,130, filed 1/21/2000. The Pause instruction allows a thread of execution to be temporarily suspended either until a count is reached or until an instruction has passed through the processor pipeline. Different techniques may be useful in allowing programmers to more efficiently harness the resources of a multi-threaded processor.

Brief Description of the Figures

[0006] The present invention is illustrated by way of example and not limitation in the Figures of the accompanying drawings.

[0007] Figure 1 illustrates one embodiment of a multi-threaded processor having a monitor to monitor memory accesses.

[0008] Figure 2 is a flow diagram illustrating operation of the multi-threaded processor of Figure 1 according to one embodiment.

[0009] Figure 3 illustrates further details of one embodiment of a multi-threading processor.

[0010] Figure 4 illustrates resource patititioning, sharing, and duplication according to one embodiment.

[0011] Figure 5 is a flow diagram illustrating suspending and resuming execution of a thread according to one embodiment.

[0012] Figure 6a is a flow diagram illustrating activation and operation of monitoring logic according to one embodiment.

[0013] Figure 6b is a flow diagram illustrating enhancement of the observability of writes according to one embodiment.

[0014] Figure 7 is a flow diagram illustrating monitor operations according to one embodiment.

[0015] Figure 8 illustrates a system according to one embodiment.

[0016] Figures 9a – 9c illustrate various embodiments of software sequences utilizing disclosed processor instructions and techniques.

[0017] Figure 10 illustrates an alternative embodiment which allows a monitored address to remain cached.

[0018] Figure 11 illustrates various design representations or formats for simulation, emulation, and fabrication of a design using the disclosed techniques.

Detailed Description

[0019] The following description describes techniques for suspending execution of a thread until a specified memory access occurs. In the following description, numerous specific details such as logic implementations, opcodes, means to specify operands, resource partitioning/sharing/duplication implementations, types and interrelationships of system components, and logic partitioning/integration choices are set forth in order to provide a more thorough understanding of the present invention. It will be appreciated, however, by one skilled in the art that the invention may be practiced without such specific details. In other instances, control structures, gate level circuits and full software instruction sequences have not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement appropriate functionality without undue experimentation.

[0020] The disclosed techniques may allow a programmer to implement a waiting mechanism in one thread while letting other threads harness processing resources. A monitor may be set up such that a thread may be suspended until a particular memory access such as a write to a specified memory location occurs. Thus, a thread may be resumed upon a specified event without executing a processor-resource-wasting routine like a spin-wait loop. In some embodiments, partitions previously dedicated to the suspended thread may be relinquished while the thread is suspended. These and/or other disclosed techniques may advantageously improve overall processor throughput.

[0021] Figure 1 illustrates one embodiment of a multi-threaded processor 100 having a memory access monitor 110 to monitor memory accesses. A "processor" may be formed as a single integrated circuit in some embodiments. In other embodiments, multiple

integrated circuits may together form a processor, and in yet other embodiments, hardware and software routines (e.g., binary translation routines) may together form the processor. In the embodiment of Figure 1, a bus/memory controller 120 provides instructions for execution to a front end 130. The front end 130 directs the retrieval of instructions from various threads according to instruction pointers 170. Instruction pointer logic is replicated to support multiple threads.

[0022] The front end 130 feeds instructions into thread partitionable resources 140 for further processing. The thread partitionable resources 140 include logically separated partitions dedicated to particular threads when multiple threads are active within the processor 100. In one embodiment, each separate partition only contains instructions from the thread to which that portion is dedicated. The thread partitionable resources 140 may include, for example, instruction queues. When in a single thread mode, the partitions of the thread partitionable resources 140 may be combined to form a single large partition dedicated to the one thread.

[0023] The processor 100 also includes replicated state 180. The replicated state 180 includes state variables sufficient to maintain context for a logical processor. With replicated state 180, multiple threads can execute without competition for state variable storage. Additionally, register allocation logic may be replicated for each thread. The replicated state-related logic operates with the appropriate resource partitions to prepare incoming instructions for execution.

[0024] The thread partitionable resources 140 pass instructions along to shared resources 150. The shared resources 150 operate on instructions without regard to their origin. For example, scheduler and execution units may be thread-unaware shared resources. The partitionable resources 140 may feed instructions from multiple threads to the shared resources 150 by alternating between the threads in a fair manner that provides continued

progress on each active thread. Thus, the shared resources may execute the provided instructions on the appropriate state without concern for the thread mix.

[0025] The shared resources 150 may be followed by another set of thread partitionable resources 160. The thread partitionable resources 160 may include retirement resources such as a re-order buffer and the like. Accordingly, the thread partitionable resources 160 may ensure that execution of instructions from each thread concludes properly and that the appropriate state for that thread is appropriately updated.

[0026] As previously mentioned, it may be desirable to provide programmers with a technique to implement the functionality of a spin-wait loop without requiring constant polling of a memory location or even execution of instructions. Thus, the processor 100 of Figure 1 includes the memory access monitor 110. The memory access monitor 110 is programmable with information about a memory access cycle for which the monitor 110 can be enabled to watch. Accordingly, the monitor 110 includes a monitor cycle information register 112, which is compared against bus cycle information received from the bus/memory controller 120 by comparison logic 114. If a match occurs, a resume thread signal is generated to re-start a suspended thread. Memory access information may be obtained from internal and/or external buses of the processor.

[0027] The monitor cycle information register 112 may contain details specifying the type of cycle and/or the address which should trigger the resumption of a thread. In one embodiment, the monitor cycle information register 112 stores a physical address, and the monitor watches for any bus cycle that indicates an actual or potential write to that physical address. Such a cycle may be in the form of an explicit write cycle and/or may be a read for ownership or an invalidating cycle by another agent attempting to take exclusive ownership of a cacheable line so that it can write to that line without an external bus transaction. In any case, the monitor may be programmed to trigger on various

transactions in different embodiments.

[0028] The operations of the embodiment of Figure 1 may be further explained with reference to the flow diagram of Figure 2. In one embodiment, the instruction set of the processor 100 includes a MONITOR opcode (instruction) which sets up the monitor transaction information. In block 200, the MONITOR opcode is received as a part of the sequence of instructions of a first thread (T1). As indicated in block 210, in response to the MONITOR opcode, the processor 100 enables the monitor 110 to monitor memory accesses for the specified memory access. The triggering memory access may be specified by an implicit or explicit operand. Therefore, executing the MONITOR opcode may specify the monitor address as the monitor address can be stored in advance in a register or other location as an implicit operand. As indicated in block 215, the monitor tests whether the specified cycle is detected. If not, the monitor continues monitoring memory accesses. If the triggering cycle is detected, then a monitor event pending indicator is set as indicated in block 220.

[0029] The execution of the MONITOR opcode triggers the activation of the monitor 110. The monitor 110 may begin to operate in parallel with other operations in the processor. In one embodiment, the MONITOR instruction itself only sets up the monitor 110 with the proper memory cycle information and activates the monitor 110, without unmasking monitor events. In other words, in this embodiment, after the execution of the MONITOR opcode, monitor events may accrue, but may not be recognized unless they are explicitly unmasked.

[0030] Thus, in block 225, triggering of a memory wait is indicated as a separate event. In some embodiments, a memory wait (MWAIT) opcode may be used to trigger the recognition of monitor events and the suspension of T1. Using two separate instructions to set up and trigger the thread suspension may provide a programmer added flexibility

and allow more efficient programming. An alternative embodiment, however, triggers the memory wait from the first opcode which also set up the monitor 110. In either case, one or more instructions arm the monitor and enable recognition of monitor events.

[0031] In embodiments where separate opcodes are used to arm the monitor 110 and to trigger the recognition of monitor events, it may be advantageous to perform a test to ensure that the monitor has been activated before suspending the thread as shown in block 230. Additionally, by testing if a monitor event is already pending (not shown), suspension of T1 may be avoided, and operation may continue in block 250. Assuming the monitor 110 has been enabled and no monitor events are already pending, T1 may be suspended as shown in block 235.

[0032] With T1 suspended, the processor enters an implementation dependent state which allows other threads to more fully utilize the processor resources. In some embodiments, the processor may relinquish some or all of the partitions of partitionable resources 140 and 160 that were dedicated to T1. In other embodiments, different permutations of the MONITOR opcode or settings associated therewith may indicate which resources to relinquish, if any. For example, when a programmer anticipates a shorter wait, the thread may be suspended, but maintain its resource partitions. Throughput is still enhanced because the shared resources may be used exclusively by other threads during the thread suspension period. When a longer wait is anticipated, relinquishing all partitions associated with the suspended thread allows other threads to have additional resources, potentially increasing the throughput of the other threads. The additional throughput, however, comes at the cost of the overhead associated with removing and adding partitions when threads are respectively suspended and resumed.

[0033] T1 remains in a suspended state until a monitor event is pending. As previously discussed, the monitor 110 operates independently to detect and signal monitor events

(blocks 215 – 220). If the processor detects that a monitor event is pending in block 240, then T1 is resumed, as indicated in block 250. No active processing of instructions in T1 needs to occur for the monitor event to wake up T1. Rather T1 remains suspended and the enabled monitor 110 signals an event to the processor. The processor handles the event, recognizes that the event indicates T1 should be resumed, and performs the appropriate actions to resume T1.

[0034] Thus, the embodiments of Figures 1 and 2 provide techniques to allow a thread suspended by a program to be resumed upon the occurrence of a specified memory access. In one embodiment, other events also cause T1 to be resumed. For example, an interrupt may cause T1 to resume. Such an implementation advantageously allows the monitor to be less than perfect in that it may miss (not detect) certain memory accesses or other conditions that should cause the thread to resume. As a result, T1 may be awakened unnecessarily at times. However, such an implementation reduces the likelihood that T1 will become permanently frozen due to a missed event, simplifying hardware design and validation. The unnecessary awakenings of T1 may be only a minor inconvenience as a loop may be constructed to have T1 double-check whether the condition it was awaiting truly did occur, and if not to suspend itself once again.

[0035] In some embodiments, the thread partitionable resources, the replicated resources, and the shared resources may be arranged differently. In some embodiments, there may not be partitionable resources on both ends of the shared resources. In some embodiments, the partitionable resources may not be strictly partitioned, but rather may allow some instructions to cross partitions or may allow partitions to vary in size depending on the thread being executed in that partition or the total number of threads being executed. Additionally, different mixes of resources may be designated as shared, duplicated, and partitioned resources.

[0036] Figure 3 illustrates further details of one embodiment of a multi-threading processor. The embodiment of Figure 3 includes coherency related logic 350, one implementation of a monitor 310, and one specific implementation of thread suspend and resume logic 377, among other things. In the embodiment of Figure 3, a bus interface 300 includes a bus controller 340, event detect logic 345, a monitor 310, and the coherency related logic 350.

[0037] The bus interface 300 provides instructions to a front end 365, which performs micro-operand (uOP) generation, generating uOPs from macroinstructions. Execution resources 370 receive uOPs from the front end 365, and back end logic 380 retires the various uOPs after they are executed. In one embodiment, out-of-order execution is supported by the front end, back end, and execution resources.

[0038] Various details of operations are further discussed with respect to Figures 5-9. Briefly, however, a MONITOR opcode may enter the processor through the bus interface 300 and be prepared for execution by the front end 365. In one embodiment, a special MONITOR uOP is generated for execution by the execution resources 370. The MONITOR uOP may be treated similarly to a store operation by the execution units, with the monitor address being translated by address translation logic 375 into a physical address, which is provided to the monitor 310. The monitor 310 communicates with thread suspend and resume logic 377 to cause resumption of threads. The thread suspend and resume logic may perform partition and anneal resources as the number of active threads changes.

[0039] For example, Figure 4 illustrates the partitioning, duplication, and sharing of resources according to one embodiment. Partitioned resources may be partitioned and annealed (fused back together for re-use by other threads) according to the ebb and flow of active threads in the machine. In the embodiment of Figure 4, duplicated resources

include instruction pointer logic in the instruction fetch portion of the pipeline, register renaming logic in the rename portion of the pipeline, state variables (not shown, but referenced in various stages in the pipeline), and an interrupt controller (not shown, generally asynchronous to pipeline). Shared resources in the embodiment of Figure 4 include schedulers in the schedule stage of the pipeline, a pool of registers in the register read and write portions of the pipeline, execution resources in the execute portion of the pipeline. Additionally, a trace cache and an L1 data cache may be shared resources populated according to memory accesses without regard to thread context. In other embodiments, consideration of thread context may be used in caching decisions. Partitioned resources in the embodiment of Figure 4 include two queues in queuing stages of the pipeline, a re-order buffer in a retirement stage of the pipeline, and a store buffer. Thread selection multiplexing logic alternates between the various duplicated and partitioned resources to provide reasonable access to both threads.

[0040] For exemplary purposes, it is assumed that the partitioning, sharing, and duplication shown in Figure 4 is utilized in conjunction with the embodiment of Figure 3 in further describing operation of an embodiment of the processor of Figure 3. In particular, further details of operation of the embodiment of Figure 3 will now be discussed with respect to the flow diagram of Figure 5. The processor is assumed to be executing in a multi-threading mode, with at least two threads active.

[0041] In block 500, the front end 365 receives a MONITOR opcode during execution of a first thread (T1). A special monitor uOP is generated by the front end 365 in one embodiment. The MONITOR uOP is passed to the execution resources 370. The monitor uOP has an associated address which indicates the address to be monitored (the monitor address). The associated address may be in the form of an explicit operand or an implicit operand (i.e., the associated address is to be taken from a predetermined register or other

storage location). The associated address "indicates" the monitor address in that it conveys enough information to determine the monitor address (possibly in conjunction with other registers or information). For example, the associated address may be a linear address which has a corresponding physical address that is the appropriate monitor address. Alternatively, the monitor address could be given in virtual address format, or could be indicated as a relative address, or specified in other known or convenient address-specifying manners. If virtual address operands are used, it may be desirable to allow general protection faults to be recognized as break events.

[0042] The monitor address may indicate any convenient unit of memory for monitoring. For example, in one embodiment, the monitor address may indicate a cache line. However, in alternative embodiments, the monitor address may indicate a portion of a cache line, a specific/selected size portion or unit of memory which may bear different relationships to the cache line sizes of different processors, or a singe address. The monitor address thus may indicate a unit that includes data specified by the operand (and more data) or may indicate specifically an address for a desired unit of data.

[0043] In the embodiment of Figure 3, the monitor address is provided to the address translation logic 375 and passed along to the monitor 310, where it is stored in a monitor address register 335. In response to the MONITOR opcode, the execution resources 370 then enable and activate the monitor 310 as indicated in block 510 and further detailed in Figure 6. As will be further discussed below with respect to Figure 6, it may be advantageous to fence any store operations that occur after the MONITOR opcode to ensure that stores are processed and therefore detected before any thread suspension occurs. Thus, some operations may need to occur as a result of activating the monitor 310 before any subsequent instructions can be undertaken in this embodiment. However, block 510 is shown as occurring in parallel with block 505 because the monitor 310

continues to operate in parallel with other operations until a break event occurs once it is activated by the MONITOR opcode in this embodiment.

[0044] In block 505, a memory wait (MWAIT) opcode is received in thread 1, and passed to execution. Execution of the MWAIT opcode unmasks monitor events in the embodiment of Figure 5. In response to the MWAIT opcode, a test is performed, as indicated in block 515, to determine whether a monitor event is pending. If no monitor event is pending, then a test is performed in block 520 to ensure that the monitor is active. For example, the if an MWAIT is executed without previously executing a MONITOR, the monitor 310 would not be active. If either the monitor is inactive or a monitor event is pending, then thread 1 execution is continued in block 580.

[0045] If the monitor 310 is active and no monitor event is pending, then thread 1 execution is suspended as indicated in block 525. The thread suspend/resume logic 377 includes pipeline flush logic 382, which drains the processor pipeline in order to clear all instructions as indicated in block 530. Once the pipeline has been drained, partition/anneal logic 385 causes any partitioned resources associated exclusively with thread 1 to be relinquished for use by other threads as indicated in block 535. These relinquished resources are annealed to form a set of larger resources for the remaining active threads to utilize. For example, referring to the two thread example of Figure 4, all instructions related to thread 1 are drained from both queues. Each pair of queues is then combined to provide a larger queue to the second thread. Similarly, more registers from the register pool are made available to the second thread, more entries from the store buffer are freed for the second thread, and more entries in the re-order buffer are made available to the second thread. In essence, these structures are returned to single dedicated structures of twice the size. Of course, different proportions may result from implementations using different numbers of threads.

[0046] In blocks 540, 545, and 550, various events are tested to determine whether thread 1 should be resumed. Notably, these tests are not performed by instructions being executed as a part of thread 1. Rather, these operations are performed by the processor in parallel to its processing of other threads. As will be discussed in further detail with respect to Figure 6, the monitor itself checks whether a monitor write event has occurred and so indicates by setting an event pending indicator. The event pending indicator is provided via an EVENT signal to the suspend/resume logic 377 (e.g., microcode). Microcode may recognize the monitor event at an appropriate instruction boundary in one embodiment (block 540) since this event was unmasked by the MWAIT opcode in block 505. Event detect logic 345 may detect other events, such as interrupts, that are designated as break events (block 545). Additionally, an optional timer may be used periodically exit the memory wait state to ensure that the processor does not become frozen due to some particular sequence of events (block 550). If none of these events signal an exit to the memory wait state, then thread 1 remains suspended.

[0047] If thread 1 is resumed, the thread/suspend resume logic 377 is again activated upon detection of the appropriate event. Again, the pipeline is flushed, as indicated in block 560, to drain instructions from the pipeline so that resources can be once again partitioned to accommodate the soon-to-be-awakened thread 1. In block 570, the appropriate resources are re-partitioned, and thread 1 is resumed in block 580.

[0048] Figure 6a illustrates further details of the activation and operation of the monitor 310. In block 600, the front end fetching for thread 1 is stopped to prevent further thread 1 operations from entering the machine. In block 605, the associated address operand is converted from being a linear address to a physical address by the address translation logic 375. In block 610, the observability of writes to the monitored address are increased. In general, the objective of this operation is to force caching agents to make write operations

which would affect the information stored at the monitor address visible to the monitor 310 itself. More details of one specific implementation are discussed with respect to Figure 6b. In block 615, the physical address for monitoring is stored, although notably this address may be stored earlier or later in this sequence.

[0049] Next, as indicated in block 620, the monitor is enabled. The monitor monitors bus cycles for writes to the physical address which is the monitor address stored in the monitor address register 335. Further details of the monitoring operation are discussed below with respect to Figure 7. After the monitor is enabled, a store fence operation is executed as indicated in block 625. The store fence helps ensure that all stores in the machine are processed at the time the MONITOR opcode completes execution. With all stores from before the MONITOR being drained from the machine, the likelihood that a memory wait state will be entered erroneously is reduced. The store fence operation, however, is a precaution, and can be a time consuming operation.

[0050] This store fence is optional because the MONITOR/MWAIT mechanism of this embodiment has been designed as a multiple exit mechanism. In other words, various events such as certain interrupts, system or on board timers, etc., may also cause exit from the memory wait state. Thus, it is not guaranteed in this embodiment that the only reason the thread will be awakened is because the data value being monitored has changed. Accordingly (see also Figure 9a-c below), in this implementation, software should double-check whether the particular value stored in memory has changed. In one embodiment, some events including assertion of INTR, NMI and SMI interrupts; machine check interrupts; and faults are break events, and others including powerdown events are not. In one embodiment, assertion of the A20M pin is also a break event.

[0051] As indicated in block 630, the monitor continues to test whether bus cycles occurring indicate or appear to indicate a write to the monitor address. If such a bus cycle

is detected, the monitor event pending indicator is set, as indicated in block 635. After execution of the MWAIT opcode (block 505, Figure 5), this event pending indicator is serviced as an event and causes thread resumption in blocks 560 – 580 of Figure 5. Additionally, events that change address translation may cause thread 1 to resume. For example, events that cause a translation look-aside buffer to be flushed may trigger resumption of thread 1 since the translation made to generate the monitor address from a linear to a physical address may no longer be valid. For example, in an x86 Intel Architecture compatible processor, writes to control registers CR0, CR3 and CR4, as well as certain machine specific registers may cause exit of the memory wait state.

[0052] As noted above, Figure 6b illustrates further details of the enhancement of observability of write to the monitor address (block 610 in Fig. 6a). In one embodiment, the processor flushes the cache line associated with the monitor address from all internal caches of the processor as indicated in block 650. As a result of this flushing, any subsequent write to the monitor address reaches the bus interface 300, allowing detection by the monitor 310 which is included in the bus interface 300. In one embodiment, the MONITOR uOP is modeled after and has the same fault model as a cache line flush CLFLUSH instruction which is an existing instruction in an x86 instruction set. The monitor uOP proceeds through linear to physical translation of the address, and flushing of internal caches much as CLFLUSH does; however, the bus interface recognizes the difference between MONITOR and CLFLUSH and treats the MONITOR uOP appropriately.

[0053] Next, as indicated in block 655, the coherency related logic 350 in the bus interface 300 activates read line generation logic 355 to generate a read line transaction on the processor bus. The read line transaction to the monitor address ensures that no other caches in processors on the bus store data at the monitor address in either a shared or

exclusive state (according to the well known MESI protocol). In other protocols, other states may be used; however, the transaction is designed to reduce the likelihood that another agent can write to the monitor address without the transaction being observable by the monitor 310. In other words, writes or write-indicating transactions are subsequently broadcast so they can be detected by the monitor. Once the read line operation is done, the monitor 310 begins to monitor transactions on the bus.

[0054] As additional transactions occur on the bus, the coherency related logic continues to preserve the observability of the monitor address by attempting to prevent bus agents from taking ownership of the cache line associated with the monitored address. According to one bus protocol, this may be accomplished by hit generation logic 360 asserting a HIT# signal during a snoop phase of any read of the monitor address as indicated in block 660. The assertion of HIT# prevents other caches from moving beyond the Shared state in the MESI protocol to the Exclusive and then potentially the Modified state. As a result, as indicated in block 665, no agents in the chosen coherency domain (the memory portion which is kept coherent) can have data in the modified or exclusive state (or their equivalents). The processor effectively appears to have the cache line of the monitor address cached even though it has been flushed from internal caches in this embodiment.

[0055] Referring now to Figure 7, further details of the operations associated with block 620 in Figure 6a are detailed. In particular, Figure 7 illustrates further details of operation of the monitor 310. In block 700, the monitor 310 receives request and address information from a bus controller 340 for a bus transaction. As indicated in block 710, the monitor 310 examines the bus cycle type and the address(es) affected. In particular, cycle compare logic 320 determines whether the bus cycle is a specified cycle. In one embodiment, an address comparison circuit 330 compares the bus transaction address to the monitor address stored in the monitor address register 335, and write detect logic 325

decodes the cycle type information from the bus controller 340 to detect whether a write has occurred. If a write to the monitor address occurs, a monitor event pending indicator is set as indicated in block 720. A signal (WRITE DETECTED) is provided to the thread suspend/resume logic 377 to signal the event (and will be serviced assuming it has been enabled by executing MWAIT). Finally, the monitor 310 is halted as indicated in block 730. Halting the monitor saves power, but is not critical as long as false monitor events are masked or otherwise not generated. The monitor event indicator may also be reset at this point. Typically, servicing the monitor event also masks the recognition of further monitor events until MWAIT is again executed.

[0056] In the case of a read to the monitor address, the coherency related logic 350 is activated. As indicated in block 740, a signal (such as HIT#) is asserted to prevent another agent from gaining ownership which would allow future writes without coherency broadcasts. The monitor 310 remains active and returns to block 700 after and is unaffected by a read of the monitor address. Additionally, if a transaction is neither a read nor a write to the monitor address, the monitor remains active and returns to block 700.

[0057] In some embodiments, the MONITOR instruction is limited such that only certain types of accesses may be monitored. These accesses may be ones chosen as indicative of efficient programming techniques, or may be chosen for other reasons. For example, in one embodiment, the memory access must be a cacheable store in write-back memory that is naturally aligned. A naturally aligned element is an N bit element that starts at an address divisible by N. As a result of using naturally aligned elements, a single cache line needs to be accessed (rather than two cache lines as would be needed in the case where data is split across two cache lines) in order to write to the monitored address. As a result, using naturally aligned memory addresses may simplify bus watching.

[0058] Figure 8 illustrates one embodiment of a system that utilizes disclosed multi-

threaded memory wait techniques. In the embodiment of Figure 8, a set of N multi-threading processors, processors 805-1 through 805-N are coupled to a bus 802. In other embodiments, a single processor or a mix of multi-threaded processors and single-threaded processors may be used. In addition, other known or otherwise available system arrangements may be used. For example, the processors may be connected in a point-to-point fashion, and parts such as the memory interface may be integrated into each processor.

[0059] In the embodiment of Figure 8, a memory interface 815 coupled to the bus is coupled to a memory 830 and a media interface 820. The memory 830 contains a multi-processing ready operating system 835, and instructions for a first thread 840 and instructions for a second thread 845. The instructions 830 include an idle loop according to disclosed techniques, various versions of which are shown in Figures 9a-9c.

[0060] The appropriate software to perform these various functions may be provided in any of a variety of machine readable mediums. The media interface 820 provides an interface to such software. The media interface 820 may be an interface to a storage medium (e.g., a disk drive, an optical drive, a tape drive, a volatile memory, a non-volatile memory, or the like) or to a transmission medium (e.g., a network interface or other digital or analog communications interface). The media interface 820 may read software routines from a medium (e.g., storage medium 792 or transmission medium 795). Machine readable mediums are any mediums that can store, at least temporarily, information for reading by a machine interface. This may include signal transmissions (via wire, optics, or air as the medium) and/or physical storage media 792 such as various types of disk and memory storage devices.

[0061] Figure 9a illustrates an idle loop according to one embodiment. In block 905, the MONITOR command is executed with address 1 as its operand, the monitor address. The

MWAIT command is executed in block 910 within the same thread. As previously discussed, the MWAIT instruction causes the thread to be suspended, assuming other conditions are properly met. When a break event occurs in block 915, the routine moves on to block 920 to determine if the value stored at the monitor address changed. If the value at the monitor address did change, then execution of the thread continues, as indicated in block 922. If the value did not change, then a false wake event occurred. The wake event is false in the sense that the MWAIT was exited without a memory write to the monitor address occurring. If the value did not change, then the loop returns to block 905 where the monitor is once again set up. This loop software implementation allows the monitor to be designed to allow false wake events.

[0062] Figure 9b illustrates an alternative idle loop. The embodiment of Figure 9b adds one additional check to further reduce the chance that the MWAIT instruction will fail to catch a write to the monitored memory address. Again, the flow begins in Figure 9b with the MONITOR instruction being executed with address 1 as its operand, as indicated in block 925. Additionally, in block 930, the software routine reads the memory value at the monitor address. In block 935, the software double checks to ensure that the memory value has not changed from the value indicating that the thread should be idled. If the value has changed, then thread execution is continued, as indicated in block 952. If the value has not changed, then the MWAIT instruction is executed, as indicated in block 940. As previously discussed, the thread is suspended until a break event occurs in block 945. Again, however, since false break events are allowed, whether the value has changed is again checked in block 950. If the value has not changed, then the loop returns to once again enable the monitor to track address 1, by returning to block 925. If the value has changed, then execution of the thread continue in block 952. In some embodiments, the MONITOR instruction may not need to be executed again after a false wake event before

the MWAIT instruction is executed to suspend the thread again.

[0063] Figure 9c illustrates another example of a software sequence utilizing MONITOR and MWAIT instructions. In the example of Figure 9c, the loop does not idle unless two separate tasks within the thread have no work to do. A constant value CV1 is stored in work location WL1 when there is work to be done by a first routine. Similarly, a second constant value CV2 is stored in WL2 when there is work to be done by a second routine. In order to use a single monitor address, WL1 and WL2 are chosen to be memory locations in the same cache line. Alternatively, a single work location may also be used to store status indicators for multiple tasks. For example, one or more bits in a single byte or other unit may each represent a different task.

[0064] As indicated in block 955, the monitor is set up to monitor WL1. In block 960, it is tested whether WL1 stores the constant value indicating that there is work to be done. If so, the work related to WL1 is performed, as indicated in block 965. If not, in block 970, it is tested whether WL2 stores CV2 indicated that there is work to be done related to WL2. If so, the work related to WL2 is performed, as indicated in block 975. If not, the loop may proceed to determine if it is appropriate to call a power management handler in block 980. For example, if a selected amount of time has elapsed, then the logical processor may be placed in a reduced power consumption state (e.g., one of a set of "C" states defined under the Advanced Configuration and Power Interface (ACPI) Specification, Version 1.0b (or later), published February 8, 1999, available at www.acpi.info as of the filing of the present application). If so, then the power management handler is called in block 985. In any of the cases 965, 975, and 985 where there was work to be done, the thread does that work, and then loops back to make the same determinations again after setting the monitor in block 955. In an alternative embodiment, the loop back from blocks 965, 975, and 985 could be to block 960 as long

as the monitor remains active.

[0065] If no work to be done is encountered through blocks 965, 975, and 985, then the MWAIT instruction is executed as indicated in block 990. The thread suspended state caused by MWAIT is eventually exited when a break event occurs as indicated in block 995. At this point, the loop returns to block 955 to set the monitor and thereafter determine whether either WL1 or WL2 indicate that there is work to be done. If no work is to be done (e.g., in the case of a false wake up event), the loop will return to MWAIT in block 990 and again suspend the thread until a break event occurs.

[0066] Figure 10 illustrates one alternative embodiment of a processor that allows the monitor value to remain cached in the L1 cache. The processor in Figure 10 includes execution units 1005, an L1 cache 1010, and write combining buffers between the L1 cache and an inclusive L2 cache 1030. The write combining buffers 1020 include a snoop port 1044 which ensures coherency of the internal caches with other memory via operations received by a bus interface 1040 from a bus 1045. Since coherency-affecting transactions reach the write combining buffers 1020 via the snoop port 1044, a monitor may be situated at the L1 cache level and still receive sufficient information to determine when a memory write event is occurring on the bus 1045. Thus, the line of memory corresponding to the monitor address may be kept in the L1 cache. The monitor is able to detect both writes to the L1 cache from the execution units and writes from the bus 1045 via the snoop port 1044.

[0067] Another alternative embodiment supports a two operand monitor instruction. One operand indicates the memory address as previously discussed. The second operand is a mask which indicates which of a variety of events that would otherwise not break from the memory wait state should cause a break from this particular memory wait. For example, one mask bit may indicate that masked interrupts should be allowed to break the memory

wait despite the fact that the interrupts are masked (e.g., allowing a wake up event even when the EFLAGS bit IF is set to mask interrupts). Presumably, then one of the instructions executed after the memory wait state is broken unmasks that interrupt so it is serviced. Other events that would otherwise not break the memory wait state can be enabled to break the memory wait, or conversely events that normally break the memory wait state can be disabled. As discussed with the first operand, the second operand may be explicit or implicit.

Figure 11 illustrates various design representations or formats for simulation, emulation, and fabrication of a design using the disclosed techniques. Data representing a design may represent the design in a number of manners. First, as is useful in simulations, the hardware may be represented using a hardware description language or another functional description language which essentially provides a computerized model of how the designed hardware is expected to perform. The hardware model 1110 may be stored in a storage medium 1100 such as a computer memory so that the model may be simulated using simulation software 1120 that applies a particular test suite 1130 to the hardware model 1110 to determine if it indeed functions as intended. In some embodiments, the simulation software is not recorded, captured, or contained in the medium.

Additionally, a circuit level model with logic and/or transistor gates may be produced at some stages of the design process. This model may be similarly simulated, sometimes by dedicated hardware simulators that form the model using programmable logic. This type of simulation, taken a degree further, may be an emulation technique. In any case, re-configurable hardware is another embodiment that may involve a machine readable medium storing a model employing the disclosed techniques.

Furthermore, most designs, at some stage, reach a level of data representing the physical placement of various devices in the hardware model. In the case where

conventional semiconductor fabrication techniques are used, the data representing the hardware model may be the data specifying the presence or absence of various features on different mask layers for masks used to produce the integrated circuit. Again, this data representing the integrated circuit embodies the techniques disclosed in that the circuitry or logic in the data can be simulated or fabricated to perform these techniques.

In any representation of the design, the data may be stored in any form of a computer readable medium. An optical or electrical wave 1160 modulated or otherwise generated to transmit such information, a memory 1150, or a magnetic or optical storage 1140 such as a disc may be the medium. The set of bits describing the design or the particular part of the design are an article that may be sold in and of itself or used by others for further design or fabrication.

[0068] Thus, techniques for suspending execution of a thread until a specified memory access occurs are disclosed. While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific constructions and arrangements shown and described, since various other modifications may occur to those ordinarily skilled in the art upon studying this disclosure.

What is claimed is:

1. A processor comprising:

a plurality of execution units to allow execution of a plurality of threads, including a first thread, said first thread having a first instruction having an associated address operand indicating a monitor address;

suspend logic to suspend execution of said first thread;

a monitor to cause resumption of the first thread in response to a memory access to the monitor address.

2. The processor of claim 1 wherein said monitor is to cause resumption in response to the memory access only if the memory access indicates an actual or potential write to the monitor address.

3. The processor of claim 1 wherein the monitor is to cause resumption of the first thread in response to the memory access to the monitor address if the first thread is suspended and monitor events are unmasked.

4. The processor of claim 3 further comprising event detection logic to cause resumption of said first thread in response to an event other than said memory access.

5. The processor of claim 4 wherein said event is an interrupt.

6. The processor of claim 1 wherein said associated address operand is an implicit operand.

7. The processor of claim 6 wherein said associated address operand is stored in a predetermined register.

8. The processor of claim 1 wherein said suspend logic is to suspend execution of said first thread in response to a second instruction, wherein the first instruction enables the monitor and the second instruction unmasks events signaled by the monitor.

9 The processor of claim 8 wherein said second instruction only enables the monitor if the first instruction has been executed.

10. The processor of claim 1 wherein said suspend logic is to suspend execution of said first thread in response to the first instruction.

11. The processor of claim 8 further comprising:

coherency logic to improve visibility of stores to said monitor address.

12. The processor of claim 11 wherein said coherency logic is to ensure that no cache within a coherency domain stores information at said monitored address in a modified or exclusive state.

13. The processor of claim 12 wherein said coherency logic is to flush a cache line associated with said monitored address from any internal caches and to generate a bus read line transaction of the cache line associated with said monitor address to other processors coupled to the processor, the bus read line transaction being a multi-phase transaction provided according to a pipelined bus protocol.

14. The processor of claim 11 wherein said coherency logic is to cause said processor to generate a bus cycle to prevent any other bus agents from performing a write transaction to said monitor address without broadcasting the write transaction.

15. The processor of claim 14 further comprising bus control logic to assert a hit signal in response to another bus agent reading information at said monitor address.

16. The processor of claim 1 wherein said monitor address indicated by said associated address operand indicates one of a cache line, a portion of a cache line, or an alternatively sized unit, for data at an address indicated by said associated address operand.

17. The processor of claim 1 further comprising address translation logic to translate said associated address operand to the monitor address which is a physical address.

18. The processor of claim 1 wherein said monitor address is chosen from a set consisting of a physical address, a virtual address, a relative address, and a linear address.

19. The processor of claim 1 further comprising a plurality of partitionable resources to be partitioned to dedicate a portion of each partitionable resource to each active one of said plurality of threads when multiple threads are active, wherein said suspend logic is to relinquish any of said plurality of partitions dedicated to said first thread in response to suspending execution of said first thread.

20. The processor of claim 19 wherein the monitor is to cause said plurality of partitionable resources to be re-partitioned to accommodate execution of said first thread in response to the memory access to the monitor address.

21. The processor of claim 20 wherein said plurality of partitionable resources comprise:

an instruction queue;

a re-order buffer;

a pool of registers;

a plurality of store buffers.

22. The processor of claim 21 further comprising:

a plurality of duplicated resources, said plurality of duplicated resources being duplicated for each of said plurality of threads, said plurality of duplicated resources comprising:

a plurality of processor state variables;

an instruction pointer;

register renaming logic.

23. The processor of claim 22 further comprising:

a plurality of shared resources, said plurality of shared resources being available for use by any of said plurality of threads, said plurality of shared resources comprising:

said plurality of execution units;

a cache;

a scheduler.

24. A processor comprising:

a front end to receive a first instruction from a first thread and a second instruction from the first thread, the first instruction indicating a monitor address;

execution resources to execute the first instruction and the second instruction and to suspend execution of said first thread in response to the second instruction;

a monitor to cause resumption of the first thread in response to a memory access to the monitor address.

25. The processor of claim 24 wherein said first instruction has an operand indicating a linear address, and wherein said processor further comprises address translation logic to translate said linear address to obtain the monitor address which is a physical address.

26. The processor of claim 25 further comprising:

coherency logic to ensure that no cache in another processor coupled to the processor stores information at said monitor address in a modified or exclusive state.

27. The processor of claim 26 wherein said coherency logic is to assert a hit signal in response to another processor snooping the monitor address.

28. A processor comprising:

front end logic to receive a first instruction from a first thread, the first instruction having an associated monitor address;

a monitor coupled to receive the monitor address, and in response to said first instruction to monitor memory accesses to said monitor address and to signal an event when an access to said monitor address occurs.

29. The processor of claim 28 wherein said monitor is to signal the event in response to a write memory access that writes to the monitor address.

30. The processor of claim 28 wherein said monitor is to signal the event in response to a line invalidating transaction.

31. The processor of claim 28 further comprising:

coherency logic to ensure that no cache in another processor coupled to the processor stores information at said monitor address in a modified or exclusive state.

32. The processor of claim 31 wherein said coherency logic comprises logic to generate an internal cache flush cycle and to generate an external read line transaction.

33. The processor of claim 28 further comprising:

logic to unmask monitor events from said monitor and to suspend said first thread in response to a second instruction.

34. A processor comprising:

a plurality of execution units to execute a plurality of threads;

front end logic to receive an instruction from a first thread of said plurality of threads;

suspend logic to suspend said first thread in response to said instruction if no monitor events are pending, and to allow other ones of said plurality of threads to execute.

35. The processor of claim 34 wherein said suspend logic is to enable recognition of monitor events, including already pending monitor events.

36. The processor of claim 35 wherein said processor comprises a plurality of partitionable resources, and wherein said suspend logic is to relinquish partitions of each of said plurality of partitionable resources associated with said first thread in addition to suspending the first thread in response to the instruction.

37. A processor comprising:

a plurality of thread partitionable resources to receive instructions;

a plurality of shared resources to execute instructions in cooperation with said plurality of thread partitionable resources;

thread suspension logic to suspend a first thread in response to an instruction in said first thread, said thread suspension logic to relinquish partitions of said plurality of thread partitionable resources associated with the first thread in addition to suspending the first thread;

a monitor to cause said processor to re-partition said plurality of thread partitionable resources and to resume execution of said first thread in response

to an access to a memory address indicated by the first thread.

38. The processor of claim 37 wherein said access to the memory address is specified by a first instruction executed in said first thread and wherein the monitor is unmasked to signal monitor events to cause thread resumption by the instruction in response to which the thread suspension logic is to suspend the first thread.

39. An apparatus comprising:

means for suspending a first thread of a plurality of threads of execution;

means for detecting an access to a memory location;

means for resuming said first thread in response to the means for detecting detecting the access to the memory location.

40. The apparatus of claim 39 wherein said means for detecting the access to the memory location is enabled in response to a first instruction executed in said first thread, and wherein said means for suspending the first thread suspends the first thread in response to a second instruction executed in said first thread.

41. The apparatus of claim 40 further comprising:

coherency means for simplifying detection of the access to the memory location.

42. The apparatus of claim 41 wherein said access to the memory location is a write or an invalidating access.

43. The apparatus of claim 41 further comprising:

means for annealing resources in response to the means for suspending suspending execution of said first thread, said means for annealing freeing partitioned resources associated with said first thread for use by other ones of said plurality of threads;

means for partitioning resources for re-partitioning resources to accommodate resumption of said first thread.

44. A method comprising:

receiving a first opcode in a first thread of execution, said first opcode having an associated address operand which indicates a monitor address;

suspending said first thread;

detecting a memory access to the monitor address;

resuming said first thread in response to detecting the memory access to the monitor address.

45. The method of claim 44 wherein suspending the first thread comprises:

receiving a second instruction in the first thread;

suspending the first thread in response to the second instruction.

46. The method of claim 45 wherein the memory access is a write access.

47. The method of claim 45 further comprising translating said associated address operand into a monitored physical address, wherein detecting the memory access to the monitor address comprises detecting a write access to the monitored physical address.

48. The method of claim 44 further comprising:

preventing other agents from obtaining ownership of information stored at said monitor address.

49. The method of claim 44 wherein detecting comprises:

receiving cycle information from external bus transactions;

detecting writes to the monitor address.

50. The method of claim 44 further comprising:

resuming said first thread in response to an event other than the memory access to the monitor address.

51. The method of claim 50 wherein said event is an interrupt.

52. The method of claim 51 wherein said interrupt is a masked interrupt that is indicated by a second operand to nonetheless be considered a break event.

53. A method comprising:

receiving a first opcode executing in a first thread of execution;

translating a linear address associated with said first opcode into a physical address;

executing a bus transaction by a monitoring bus agent to ensure no other bus agent has sufficient ownership of data associated with said physical address to allow another bus agent to modify the data without informing the

monitoring bus agent;

monitoring for a write access to said physical address;

signaling a hit if another bus agent reads said physical address;

receiving a second opcode in the first thread of execution;

suspending said first thread of execution and enabling recognition of a monitor event in response to the second opcode;

resuming said first thread if the write access occurs;

resuming execution of the first thread in response to any one of a first set of events;

ignoring a second set of events.

54. The method of claim 53 wherein suspending the first thread of execution in response to the second opcode comprises:

testing whether the monitor event is pending;

testing whether a monitor is active;

if the monitor is active and no monitor event is pending, then entering a first thread suspended state.

55. The method of claim 54 wherein entering the first thread suspended state comprises:

relinquishing a plurality of registers in a register pool;

relinquishing a plurality of instruction queue entries in an instruction queue;

relinquishing a plurality of store buffer entries in a store buffer;

relinquishing a plurality of re-order buffer entries in a re-order buffer.

56. A system comprising:

a memory to store a first instruction from a first thread, the first instruction having an associated address operand indicating a monitor address;

a first processor coupled to said memory, said first processor to enable a monitor to monitor memory transactions to detect a memory access to said monitor address in response to the first instruction and to cause resumption of said first thread in response to the memory access to the monitor address.

57. The system of claim 56 wherein said memory is to store a second instruction from said first thread, and wherein said first processor is to suspend said first thread in response to the second instruction.

58. The system of claim 57 wherein said monitor is to set a monitor event pending indicator in response the memory access occurring, said monitor event pending indicator to cause said first processor to resume a thread once unmasked by said second instruction.

59. The system of claim 56 wherein said first processor includes a first cache, the system further comprising:

a second processor comprising a second cache, wherein said first processor drives a bus transaction to the second processor to force said second processor to broadcast to the first processor any transactions that allow alteration of data stored at the monitor address in the second cache.

60. The system of claim 59 wherein said first processor is to assert a signal preventing

said second processor from caching data at the monitor address in a state which would allow the second processor to modify data stored at the monitor address in the second cache without broadcasting that a modification is occurring.

61. The system of claim 60 wherein said signal indicates a cache hit and prevents the second cache from storing data at the monitor address in an exclusive state.

62. The system of claim 58 wherein said first processor is further to resume the first thread if an alternative event occurs.

63. The system of claim 62 wherein said alternative event is an interrupt.

64. The system of claim 62 wherein said first thread stored in said memory includes a loop, said loop including the first instruction and the second instruction as well as a test to determine if data at the monitor address has changed and to re-start the loop if data at the monitor address remains unchanged.

65. An article comprising a computer readable medium, which represents a processor comprising:

a plurality of execution units to allow execution of a plurality of threads, including a first thread, said first thread having a first instruction having an associated address operand indicating a monitor address;

suspend logic to suspend execution of said first thread;

a monitor to cause resumption of the first thread in response to a memory access to the monitor address.

66. The article of claim 65 wherein said monitor is to cause resumption in response to the memory access only if the memory access indicates an actual or potential write to the monitor address.

67. The article of claim 65 wherein the monitor is to cause resumption of the first thread in response to the memory access to the monitor address if the first thread is suspended and monitor events are unmasked.

68. The article of claim 65 wherein said processor further comprises event detection logic to cause resumption of said first thread in response to an event other than said memory access.

69. The article of claim 68 wherein said processor further comprises a plurality of partitionable resources to be partitioned to dedicate a portion of each partitionable resource to each active one of said plurality of threads when multiple threads are active, wherein said suspend logic is to relinquish any of said plurality of partitions dedicated to said first thread in response to suspending execution of said first thread.

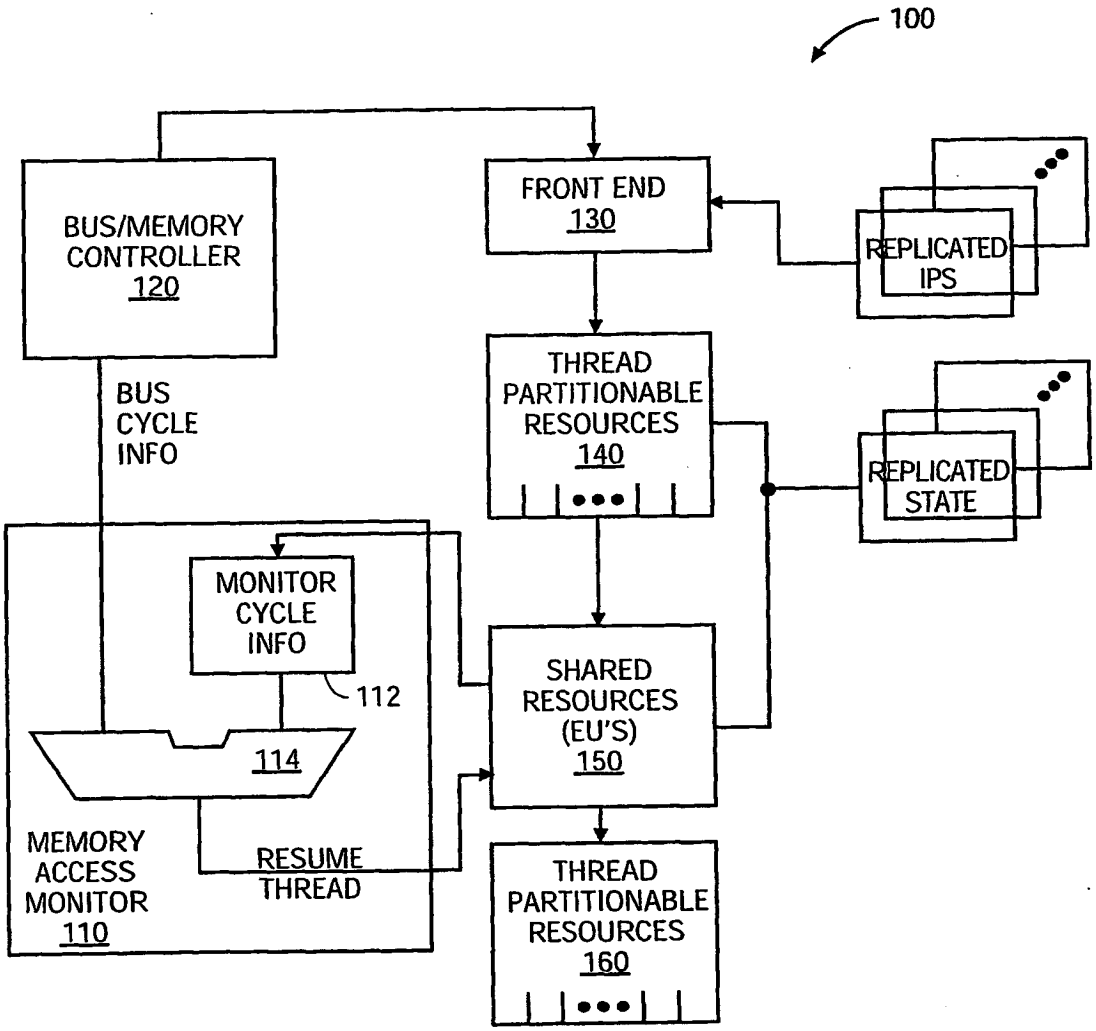
100
BUS/MEMORY CONTROLLER 120
FRONT END 130
REPLICATED IPS
BUS CYCLE INFO
THREAD PARTITIONABLE RESOURCES 140
REPLICATED STATE
MONITOR CYCLE INFO
112
114
SHARED RESOURCES (EU'S) 150
MEMORY ACCESS MONITOR 110
RESUME THREAD
THREAD PARTITIONABLE RESOURCES 160

FIG. 1

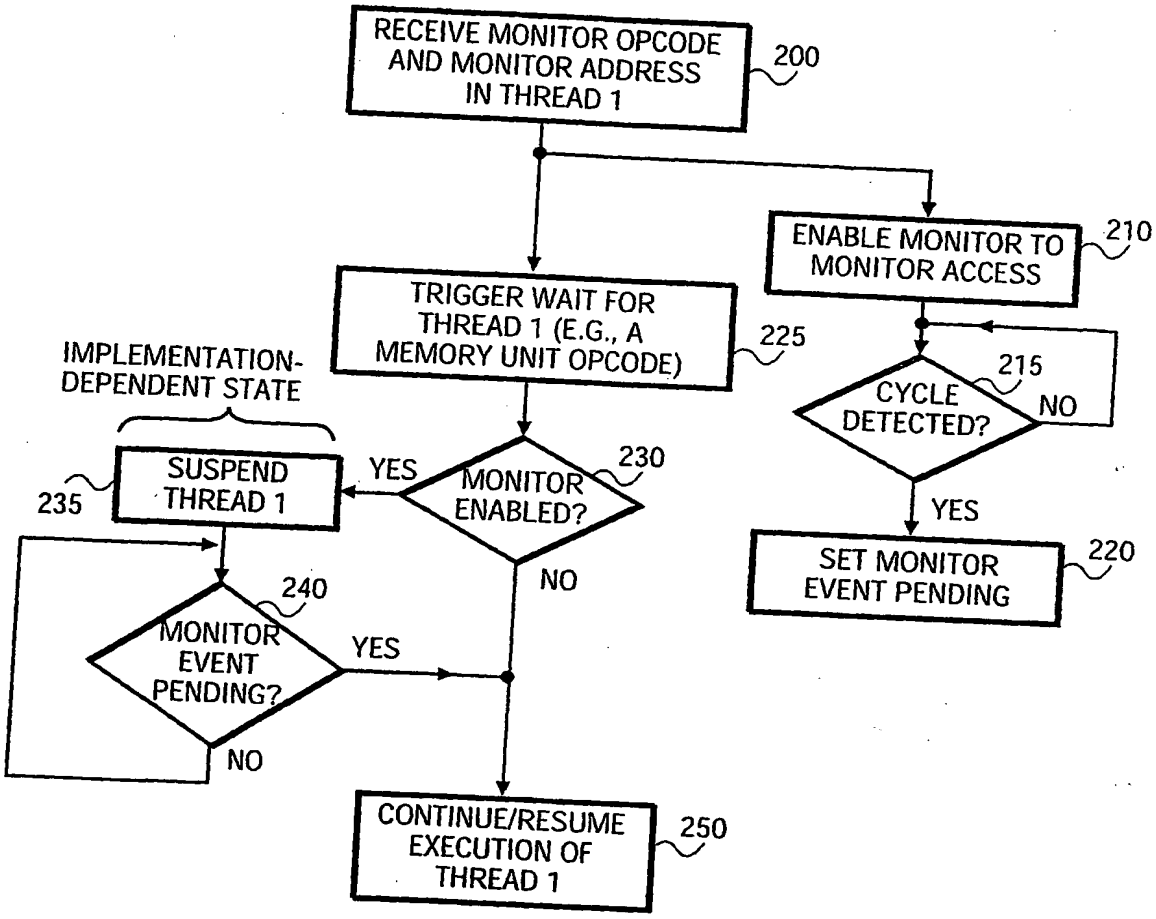
RECEIVE MONITOR OPCODE AND MONITOR ADDRESS IN THREAD 1
200
ENABLE MONITOR TO MONITOR ACCESS
210
TRIGGER WAIT FOR THREAD 1 (E.G., A MEMORY UNIT OPCODE)
225
IMPLEMENTATION-DEPENDENT STATE
CYCLE DETECTED?
215
NO
YES
SUSPEND THREAD 1
235
YES
MONITOR ENABLED?
230
SET MONITOR EVENT PENDING
220
NO
MONITOR EVENT PENDING?
240
YES
NO
CONTINUE/RESUME EXECUTION OF THREAD 1
250

FIG. 2

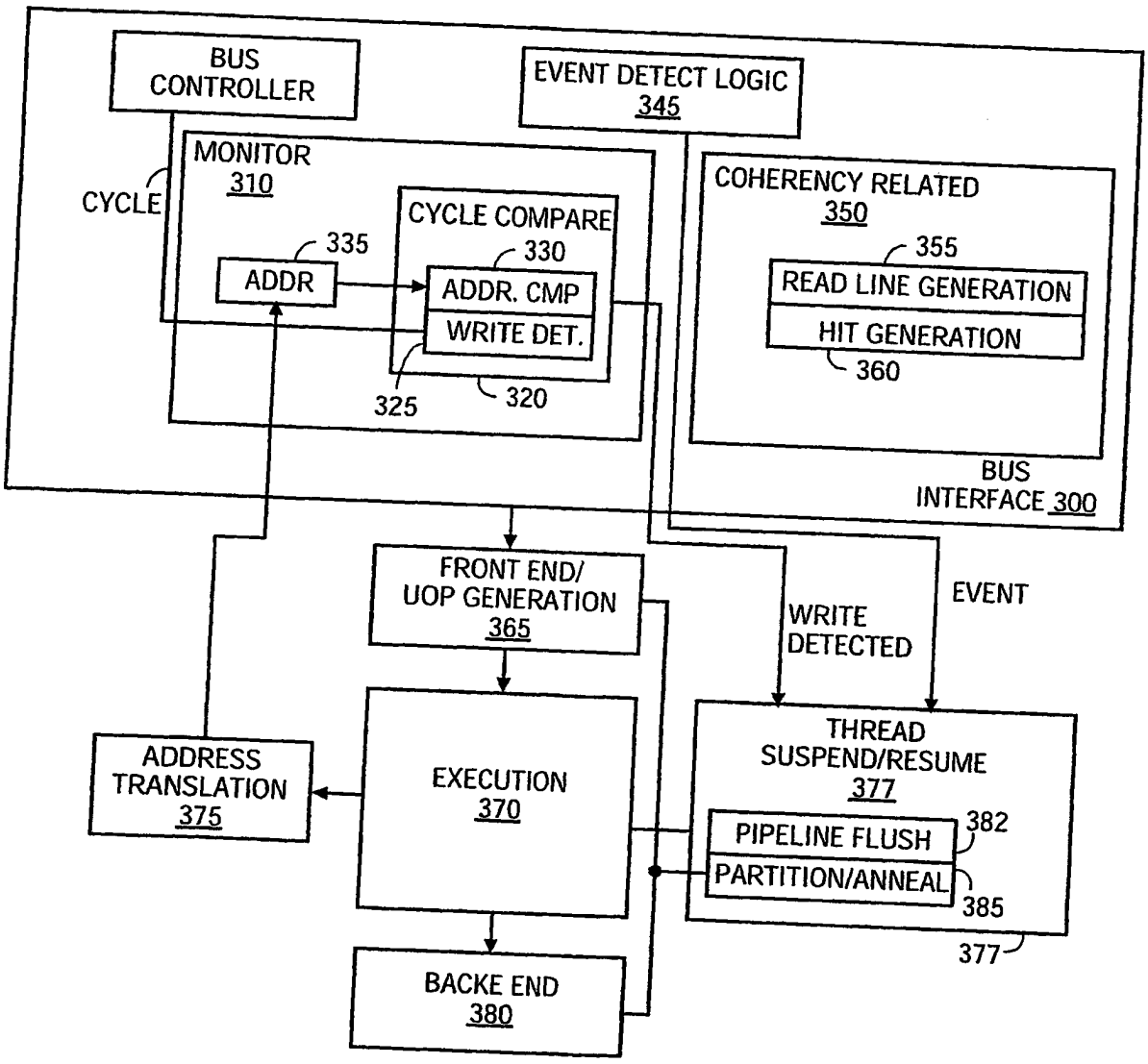
BUS CONTROLLER
EVENT DETECT LOGIC 345
MONITOR 310
CYCLE
CYCLE COMPARE
335
330
ADDR
ADDR. CMP
WRITE DET.
325
320
COHERENCY RELATED 350
355
READ LINE GENERATION
HIT GENERATION
360
BUS INTERFACE 300
FRONT END/ UOP GENERATION 365
WRITE DETECTED
EVENT
ADDRESS TRANSLATION 375
EXECUTION 370
THREAD SUSPEND/RESUME 377
PIPELINE FLUSH
382
PARTITION/ANNEAL
385
377
BACKE END 380

FIG. 3

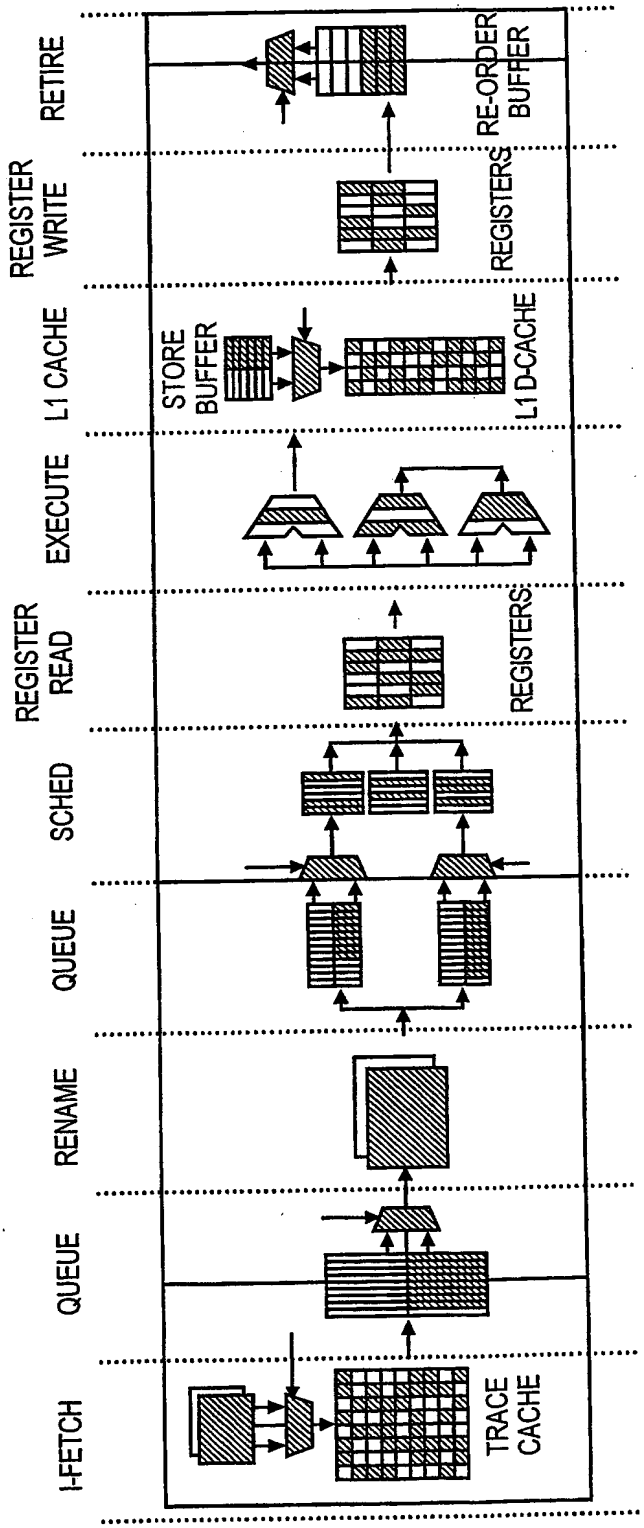
I-FETCH
QUEUE
RENAME
QUEUE
SCHED
REGISTER READ
EXECUTE
L1 CACHE
REGISTER WRITE
RETIRE
TRACE CACHE
REGISTERS
STORE BUFFER
L1 D-CACHE
REGISTERS
RE-ORDER BUFFER

FIG. 4

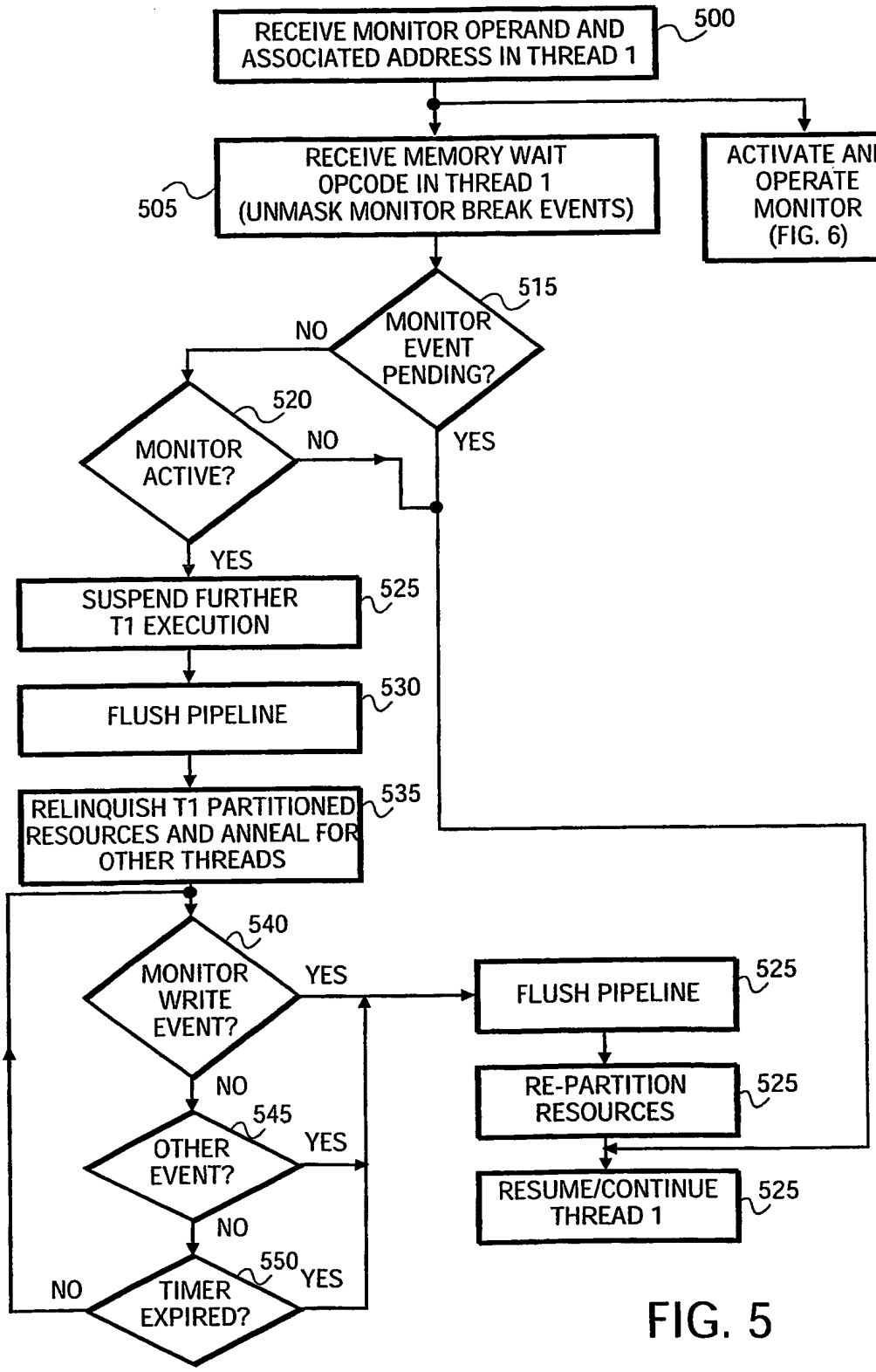
RECEIVE MONITOR OPERAND AND ASSOCIATED ADDRESS IN THREAD 1
500
RECEIVE MEMORY WAIT OPCODE IN THREAD 1 (UNMASK MONITOR BREAK EVENTS)
505
ACTIVATE AND OPERATE MONITOR (FIG. 6)
510
MONITOR EVENT PENDING?
515
NO
YES
MONITOR ACTIVE?
520
NO
YES
SUSPEND FURTHER T1 EXECUTION
525
FLUSH PIPELINE
530
RELINQUISH T1 PARTITIONED RESOURCES AND ANNEAL FOR OTHER THREADS
535
MONITOR WRITE EVENT?
540
YES
NO
FLUSH PIPELINE
525
RE-PARTITION RESOURCES
525
RESUME/CONTINUE THREAD 1
525
OTHER EVENT?
545
YES
NO
TIMER EXPIRED?
550
YES
NO

FIG. 5

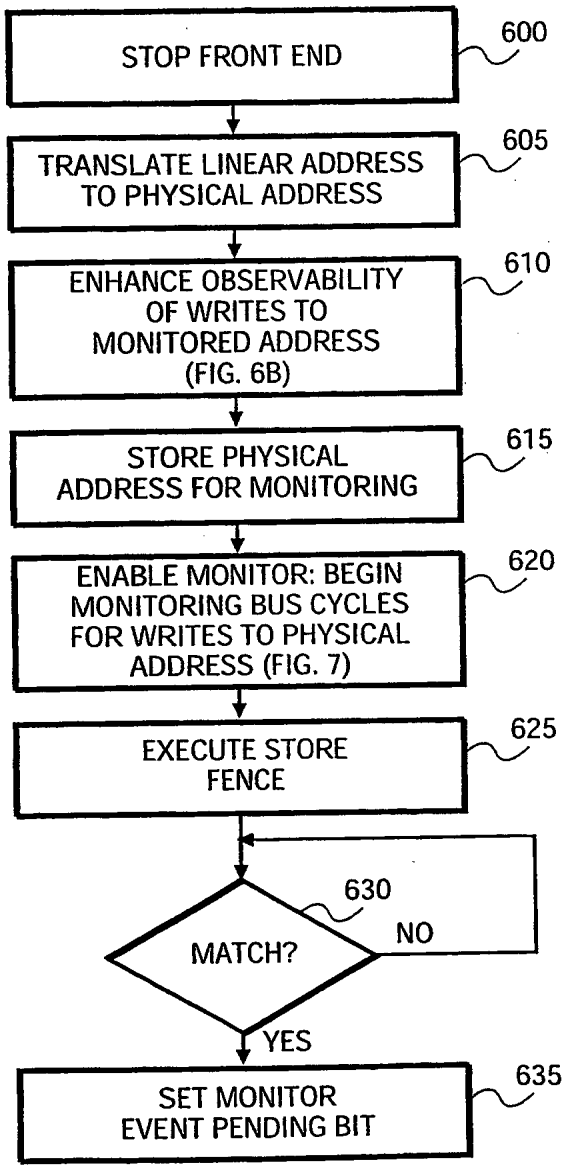

FIG. 6A

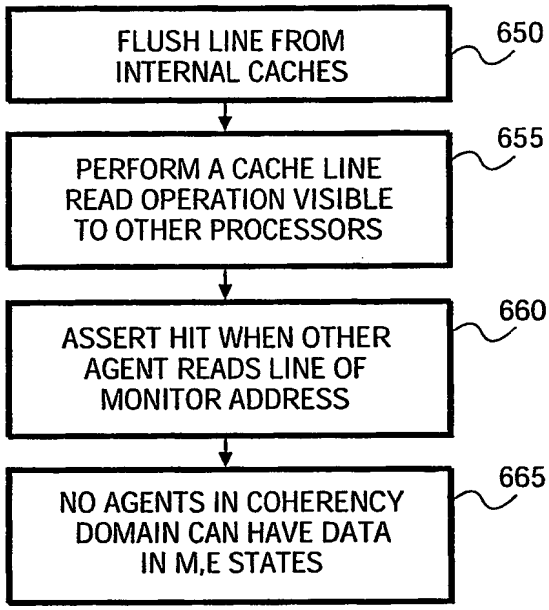
FLUSH LINE FROM INTERNAL CACHES
650
PERFORM A CACHE LINE READ OPERATION VISIBLE TO OTHER PROCESSORS
655
ASSERT HIT WHEN OTHER AGENT READS LINE OF MONITOR ADDRESS
660
NO AGENTS IN COHERENCY DOMAIN CAN HAVE DATA IN M,E STATES
665

FIG. 6B

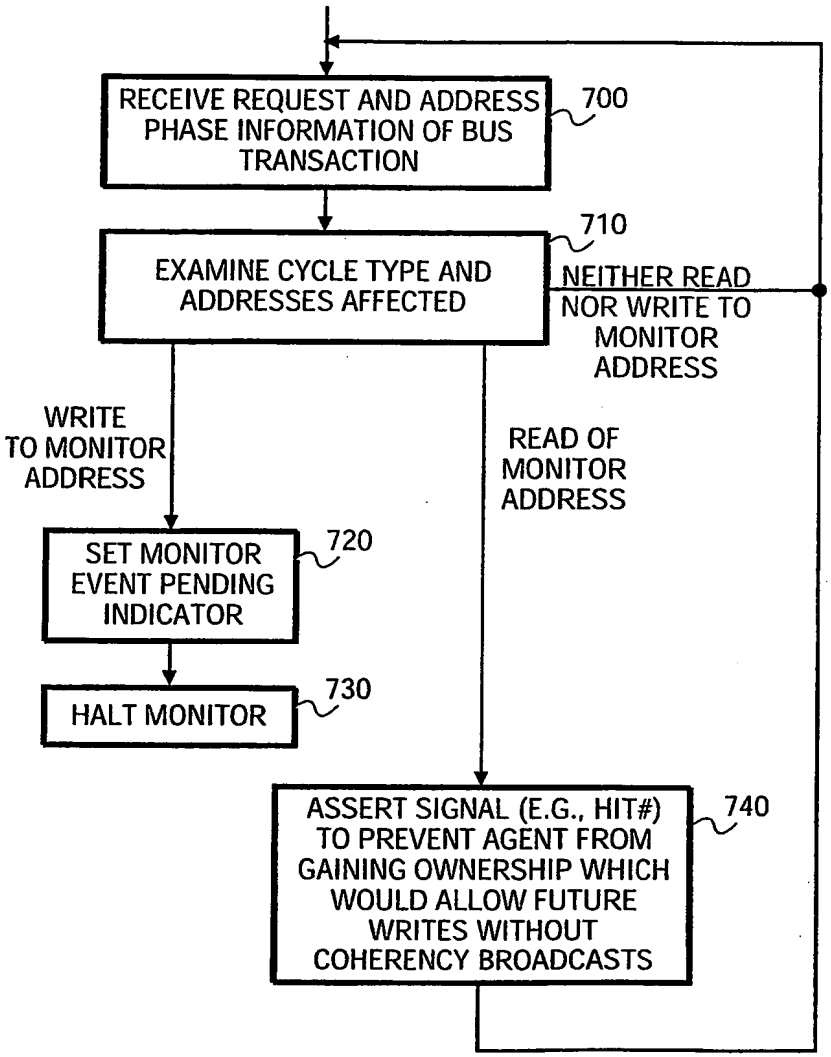

FIG. 7

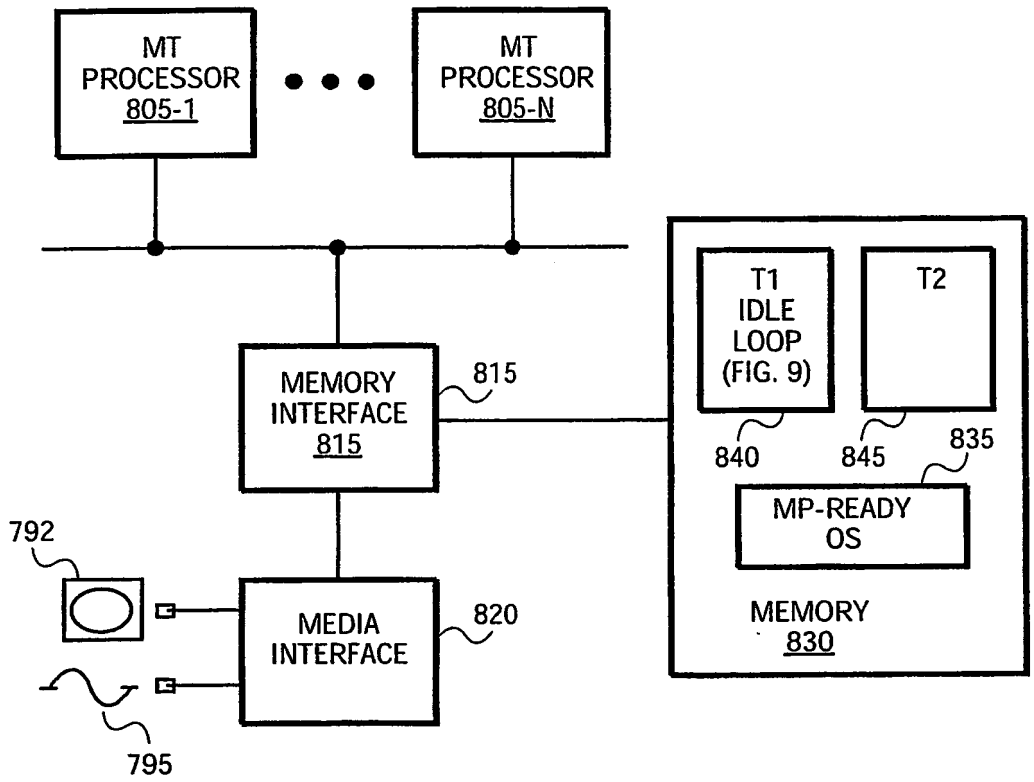
MT PROCESSOR 805-1
MT PROCESSOR 805-N
MEMORY INTERFACE 815
815
T1 IDLE LOOP (FIG. 9)
T2
840
845
835
MP-READY OS
MEMORY 830
792
MEDIA INTERFACE
820
795

FIG. 8

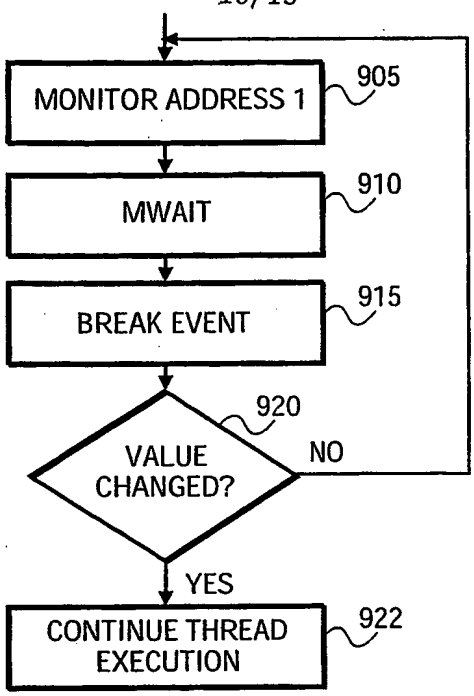
MONITOR ADDRESS 1
905
MWAIT
910
BREAK EVENT
915
VALUE CHANGED?
920
NO
YES
CONTINUE THREAD EXECUTION
922

FIG. 9A

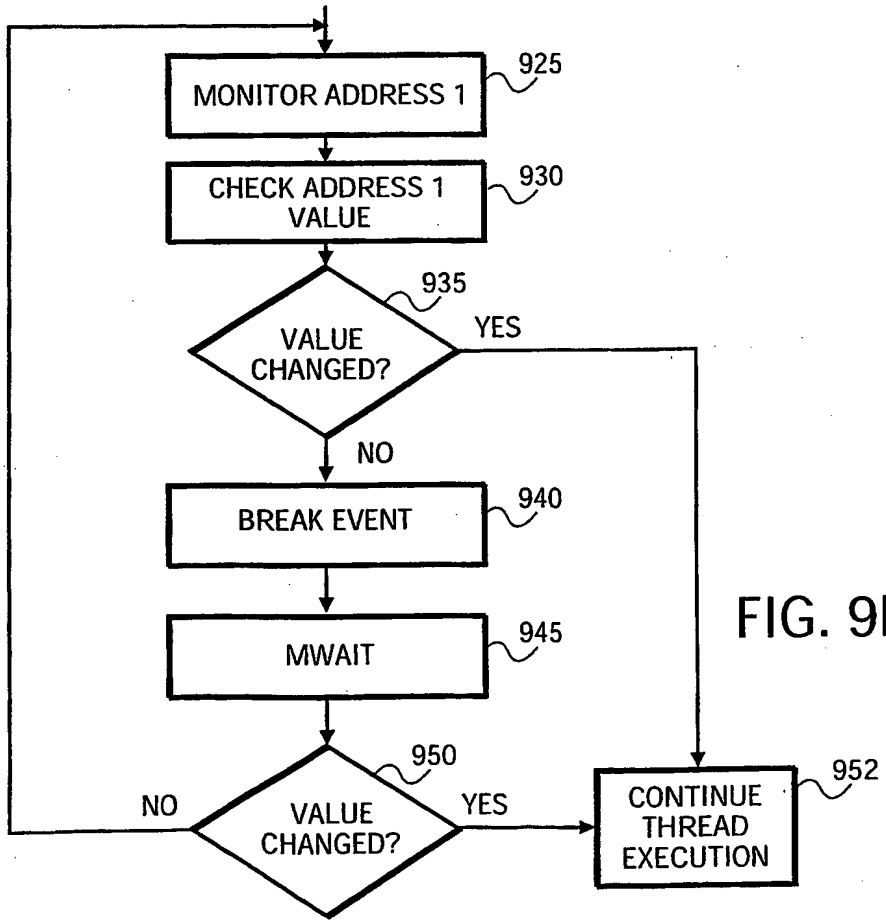
MONITOR ADDRESS 1
925
CHECK ADDRESS 1 VALUE
930
VALUE CHANGED?
935
YES
NO
BREAK EVENT
940
MWAIT
945
VALUE CHANGED?
950
NO
YES
CONTINUE THREAD EXECUTION
952

FIG. 9B

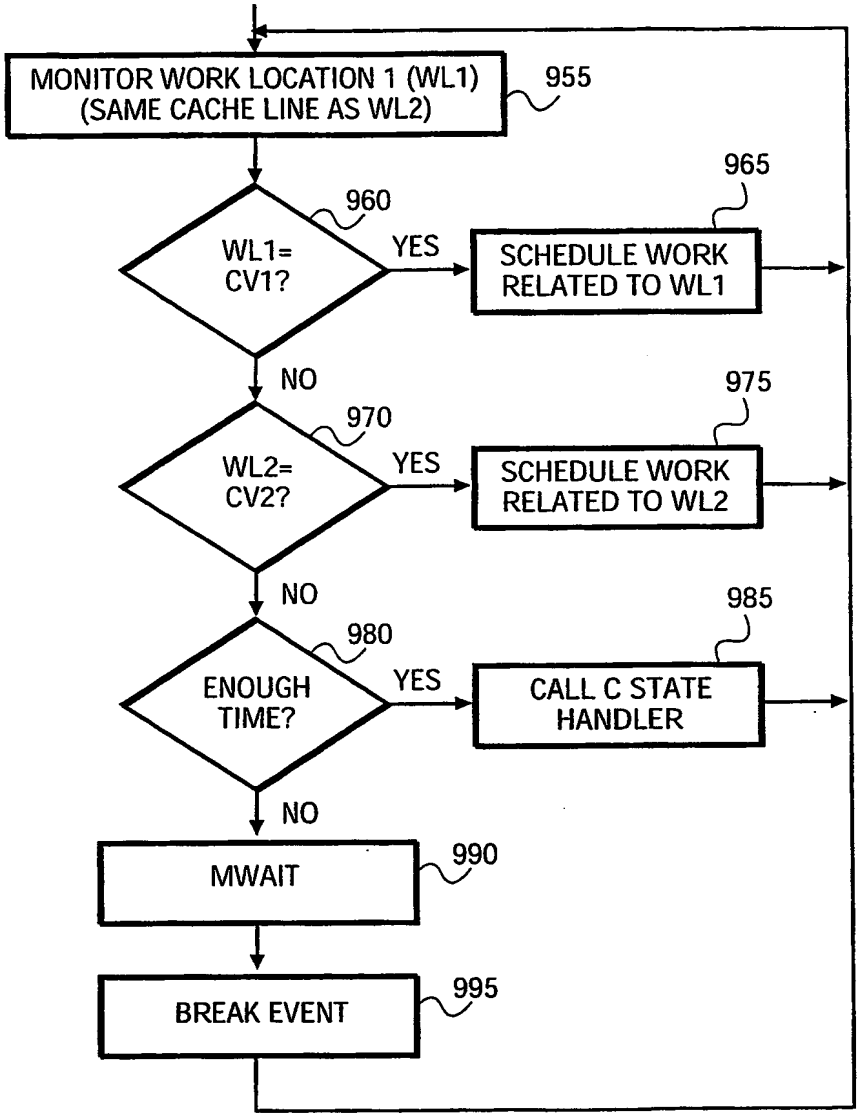
MONITOR WORK LOCATION 1 (WL1)
(SAME CACHE LINE AS WL2)
955
WL1=
CV1?
960
YES
SCHEDULE WORK
RELATED TO WL1
965
NO
WL2=
CV2?
970
YES
SCHEDULE WORK
RELATED TO WL2
975
NO
ENOUGH
TIME?
980
YES
CALL C STATE
HANDLER
985
NO
MWAIT
990
BREAK EVENT
995

FIG. 9C

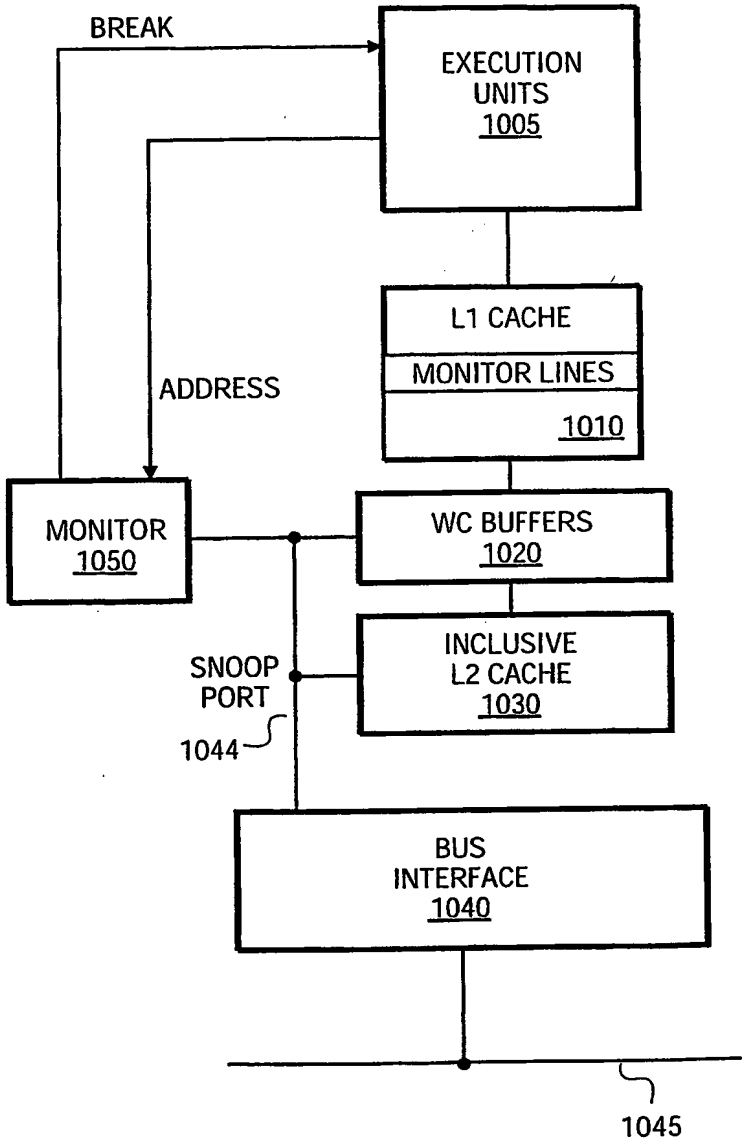
BREAK
EXECUTION UNITS 1005
L1 CACHE
MONITOR LINES
1010
ADDRESS
MONITOR 1050
WC BUFFERS 1020
INCLUSIVE L2 CACHE 1030
SNOOP PORT
1044
BUS INTERFACE 1040
1045

FIG. 10

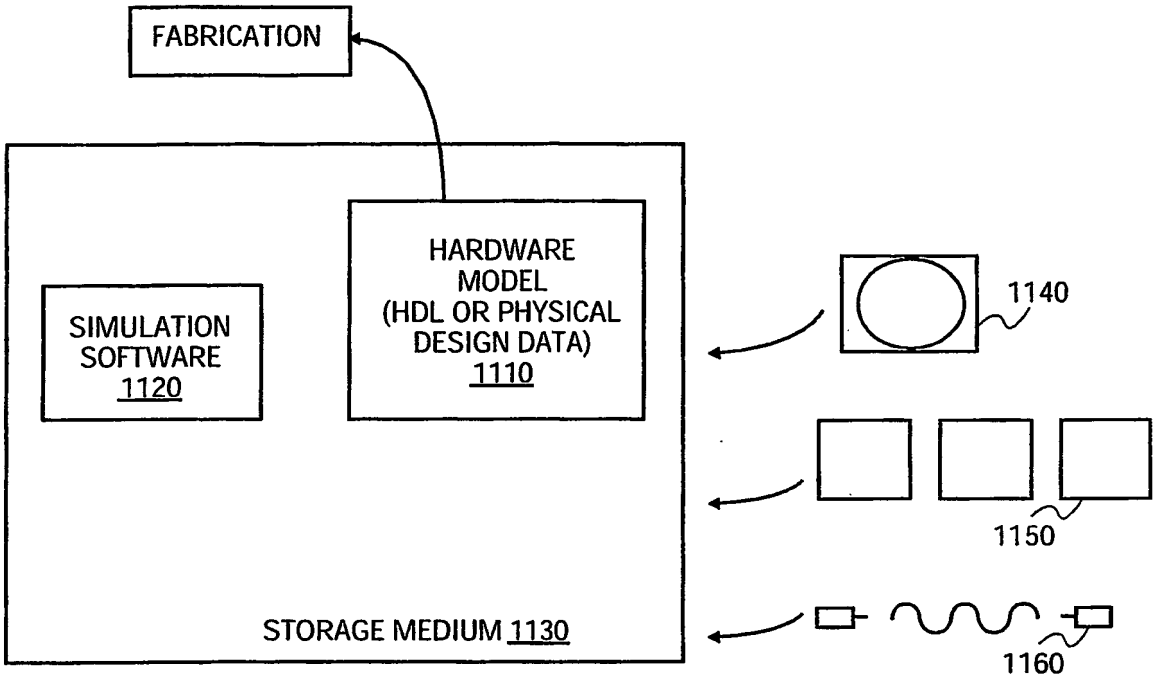
FABRICATION
HARDWARE MODEL (HDL OR PHYSICAL DESIGN DATA) 1110
SIMULATION SOFTWARE 1120
STORAGE MEDIUM 1130
1140
1150
1160

FIG. 11

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 July 2003 (17.07.2003)

PCT

(10) International Publication Number
WO 2003/058447 A3

(51) International Patent Classification[7]: G06F 9/46, 9/38, 12/08

(21) International Application Number:
PCT/US2002/039786

(22) International Filing Date:
11 December 2002 (11.12.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/039,579 31 December 2001 (31.12.2001) US

(71) Applicant: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventors: **MARR, Deborah**; 2564 NW Pettygrove Street, Portland, OR 97210 (US). **RODGERS, Scott**; 452 SW Brookwood Avenue, Hillsboro, OR 97123 (US). **HILL, David**; 37000 SW Goddard Road, Cornelius, OR 97113 (US). **KAUSHIK, Shivnandan**; 15417 NW Blakely Lane, Portland, OR 97229 (US). **CROSSLAND, James**; 10744 NW Davidson Rad, Banks, OR 97106 (US). **KOUFATY, David**; 6030 NW 163rd Place, Portland, OR 97229 (US).

(74) Agents: **MALLIE, Michael, J.** et al.; Blakely, Sokoloff, Taylor & Zafman, 7th Floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report:
16 December 2004

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A METHOD AND APPARATUS FOR SUSPENDING EXECUTION OF A THREAD UNTIL A SPECIFIED MEMORY ACCESS OCCURS

(57) Abstract: Techniques for suspending execution of a thread until a specified memory access occurs. In one embodiment, a processor includes multiple execution units capable of executing multiple threads. A first thread includes an instruction that specifies a monitor address. Suspend logic suspends execution of the first thread, and a monitor causes resumption of the first thread in response to an access to the specified monitor address.

WO 2003/058447 A3

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 02/39786

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/46 G06F9/38 G06F12/08

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, COMPENDEX, PAJ, IBM-TDB, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,X	US 6 493 741 B1 (JOERG CHRISTOPHER F ET AL) 10 December 2002 (2002-12-10) abstract column 3, line 24 - column 4, line 14 column 5, line 6 - column 6, line 11 column 6, line 55 - column 9, line 35 figures 2,5-7 ----- -/--	1-7,10, 16-18, 25,28, 29,34, 35,38, 39,44, 46,47, 50-52, 56,58, 62,63, 65-68

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
30 March 2004	22.10.2004

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	de Man, A

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 02/39786

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	ZILLES ET AL: "Time-Shifted Modules: Exploiting Code Modularity for Fine Grain Parallelization" UNIVERSITY OF WISCONSIN TECHNICAL REPORT, [Online] no. TR1430, October 2001 (2001-10), pages 1-21, XP002269277 Retrieved from the Internet: URL:http://www.cs.wisc.edu/~zilles/papers/tsm-tr1430.pdf> [retrieved on 2004-01-20] page 10	1,2,6,7, 10,16, 28,29, 34,39, 44,46, 56,65,66
Y		3-5,17, 18,25, 35,38, 47, 50-52, 58,62, 63,67,68

Y	US 5 530 597 A (BOWLES JAMES E ET AL) 25 June 1996 (1996-06-25) column 2, line 9 - line 18 column 5, line 32 - line 46	3-5,35, 38, 50-52, 58,62, 63,67,68

Y	BRADFORD ET AL: "Efficient Synchronization for Multithreaded Processors" WORKSHOP ON MULTITHREADED EXECUTION, ARCHITECTURE AND COMPILATION (MTEAC), [Online] 31 January 1998 (1998-01-31), - 4 February 1998 (1998-02-04) pages 1-4, XP002269278 Retrieved from the Internet: URL:http://albrecht.ecn.purdue.edu/~jbradfor/research/publications/98-MTEAC-text.ps.gz> [retrieved on 2004-02-04] abstract page 1, left-hand column, line 39 - line 49 page 2, left-hand column, line 40 - line 59	17,18, 25,47
A		1,2,28, 29,34, 39,44, 56,65,66

	-/--	

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 02/39786

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 361 176 A (IBM) 4 April 1990 (1990-04-04) abstract column 10, lines 26-37 column 14, line 9 - column 18, line 30 column 20, line 55 - column 21, line 21 -----	1-4,35, 38,50, 58,62, 67,68

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 02/39786

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1-7, 10, 16-18, 25, 28, 29, 34, 35, 38, 39, 44, 46, 47, 50-52, 56, 58, 62
63, 65-68

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-7, 10, 16-18, 25, 28, 29, 34, 35, 38, 39, 44, 46, 47, 50-52, 56, 58, 62, 63, 65-68

   Masking and unmasking of monitor events.

   ---

2. claims: 19-23, 36, 37, 43, 55, 69

   Relinquishing and re-partitioning of processor resources.

   ---

3. claims: 8, 9, 24, 33, 40, 45, 54, 57, 64

   Separate instructions for setting up the monitor and suspending the thread.

   ---

4. claims: 11-15, 26, 27, 30-32, 41, 42, 48, 49, 53, 59-61

   Use of coherency logic to implement monitoring functionality.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 02/39786

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6493741 B1	10-12-2002	US 2003105944 A1	05-06-2003
		US 2004073905 A1	15-04-2004
US 5530597 A	25-06-1996	CN 1081776 A ,B	09-02-1994
		DE 69322554 D1	28-01-1999
		DE 69322554 T2	19-08-1999
		EP 0581479 A1	02-02-1994
		JP 6075779 A	18-03-1994
EP 0361176 A	04-04-1990	US 4965718 A	23-10-1990
		DE 68922261 D1	24-05-1995
		DE 68922261 T2	02-11-1995
		EP 0361176 A2	04-04-1990
		JP 2236735 A	19-09-1990